Mx, A MIX-VALUED ALGEBRA

Michael Sinutko Jr.

Department of Defense
Post Office Box 1747
Washington, D.C. 20013
Phone: (202)694-3361

James H. Pugsley

Electrical Engineering Dept.
University of Maryland
College Park, Maryland 20742
Phone: (301)454-6862

Abstract

A combinatorial "mix-valued" algebra, denoted by "Mx," operates on variables representing multivalued signals and buses of any width (including width = 1). A bus of signals is represented as a single multivalued variable. All variables in Mx can range over values and sets of values and are not required to range over the same set. The function set proposed for Mx includes relational, set theoretic, and existential operators. The usual two-valued Boolean algebra is a subalgebra of Mx when all variables are binary and from the same set of values.

Only combinatorial memoryless topics are discussed, but sequential and memory circuits composed of Mx operations are known.

Mx is useful for compact technology-independent representation of digital systems during the design process.

1.0 Introduction

Mx is a technology-independent mix-valued algebra[9] for describing and designing digital circuits* interconnected by combinations of multivalued signals and buses of any mix of widths including width = 1. A p-wide input or output bus of q-valued signals is treated as a single multivalued variable ranging over up to q^p elements. Variables in Mx can range over dissimilar, hecerogeneous sets.

All functions in Mx must conform to the structure given in Section 2.1, but are otherwise arbitrary. Ten such Mx-functions are proposed and described. Other conforming functions can be used, but properties such as functional completeness are the responsibility of their designer.

For experienced logic designers, the AND, OR, and NOT Mx-operators are 2-valued Boolean operators when applied to binary variables over the same set. Also, the proposed symbolism and design procedure resemble those of 2-valued Boolean algebra.

Each Mx-gate g (function realizor)

a) accepts inputs and generates outputs from its own independent "reference set," denoted by r_g (a reference set may be ordered or not, as needed)

b) accepts and generates a "null variable" denoted by ø.

A reference set specifies all I/O (input and/or output) elements recognized by a Mx-gate ({0,1} is a 2-valued Boolean gate "reference set"). Mx-gates having differing reference sets can interact. A null variable (ø) has zero cardinality and ranges over the heterogeneous set of valueless members[9]. (The member instance of the null variable ø is a member of every set.) "Null" hereafter means null variable unless stated otherwise.

A ø-bearing input is interpreted in one of two ways at Mx-gate inputs depending on the gate type: a) it has vanished, needing no consideration in the gate output determination; or b) it appears to have no value. The "existential" Mx-operators implement (b) by waiting for a non-ø on all, none, or exactly one of the inputs before generating an output.**

Mx never requires more equations than 2-valued Boolean algebra to describe a binary circuit with busing because Mx-equations are written for a whole bus at a time; not for each bus signal component. Using Mx generally results in a very com-

---

* We pronounce Mx as "mix." "Circuit" or "network" means combinatorial circuit or network.

** The "waiting" behavior is similar to that found prior to the "firing" of Petri net "transitions."[1] "Transitions" are undefined for input values not within the operator value-universe. Mx operations are defined under such a condition.

pact set of equations which fully describe the intended behavior of the network. The compactness becomes dramatic with the increased use of busing.

A Mx-logic design needs no redesign to change realizations; only a new translation of the logic design into hardware. Standard implementation circuits for the Mx-gates in various technologies may evolve to fill this need.

The reader is invited to preview the application Sections 2.3 and 2.4 for samples of Mx's operability.

2.0 Formal Definition of Mx

Mx is defined as the pair (S,F) wherein

S is a totally ordered set, the union of the sets I, O, and R_g, and further decomposed such that no member of S is a set, where

I = $\{X_1, X_2, \ldots\}$ is the set of all input sets X_m where $X_m = \{x_1, x_2, \ldots, x_n\}$, a set of input variables, for arbitrary m and n,

O = $\{z_1, z_2, \ldots\}$ is the set of all output variables z_m,

$R_g = \{R_{g1}, R_{g2}, \ldots\}$ is the set of all sets of reference sets r_{gi}, and $R_{gm} = \{r_{g1}, r_{g2}, \ldots, r_{gn}\}$ for arbitrary i, m, and n, and

F is a set of functions from I to O.

——End of Definition——

Reference sets, set "I" and set "O" are formally independent of each other, but to date, meaningful applications of F have only involved I and O sets which intersect R_g.

A ø can render its sending and receiving Mx-operators temporarily non-associative and non-distributive; it implies an into map. Thus F is conditionally associative and distributive.

2.1 The Mx "General Function Structure" (GFS)

The following definition is in the Mx GFS, to which all functions in F must conform. "fi" and "ci" are function and conditions "i," respectively. Note that although $x_i \in X$, $x_i \subseteq r_g$ is used because generally, x_i can be a set which is a member of X, and a subset but not a member of r_g. x_i can be both a member of X and r_g.

$$\#(X) \equiv \begin{cases} \left\{ \begin{array}{l} \forall x_i \in X \exists : x_i \subseteq r_g, \text{ only one of:} \\ f1 \text{ iff } c1 \text{ are met,} \\ f2 \text{ iff } c2 \text{ are met,} \\ \quad \vdots \\ fn \text{ iff } cn \text{ are met.} \end{array} \right\} \\ \text{the null variable } (\emptyset) \text{ otherwise,} \end{cases}$$

where "#" is an arbitrary gate operation, X is the input set, $X = \{x_1, x_2, \ldots, x_m\}$, and $fj \subseteq r_g$ for $j \in \{1,2,\ldots,n\}$. "$\forall x_i \in X \exists : x_i \subseteq r_g$" is an input filter operation.

——End of Definition——

Let $x_i \in X$ be variables in an input set X, $i \in \{1,2,3,\ldots\}$. Define as an "alien" any input instance not a subset of r_g (i.e., an $(x_i \in X) \not\subseteq r_g$). If one or more aliens are present, the GFS's input filter causes the gate to yield a null. In the presence of a non-alien-bearing input, the GFS requires the output to be one of a set of user-devised conditional functions or a null.

2.2 The Proposed Function Set

Reasons for using a given 2-valued Boolean operation vary. For example, "x AND y" can mean greatest lower bound, product, min, intersection, existential (or, sentential[2]) truth or simultaneity, and maybe others. Similar statements can be made for OR and NOT. The following table lists some such interpretations on a 2-valued Boolean AND operation over {0,1}. Similar tables can be produced for OR and NOT.

x	y	AND (x,y)	product (x,y)	min (x,y)	intersect (x,y)	existential truth	existential simultaneity
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1

These ambiguities in 2-valued Boolean operator use suggest three function classes — AND, OR, and NOT — which are basic to the logic design procedure discussed in Section 2.3.

The proposed set of Mx-functions is F ≡ {AND(X), OR(X), NOT(X), UNION(X), INTERSECT(X), COMPLEMENT(X), ANDe(X), ORe(X), NOTe(X), EXIST(X)}. F addresses relational, set-theoretic, and existential domains. (The latter is concerned with the presence, or existence, of elements instead of their manipulation.) F members are further assigned to the three function classes as shown in the following table.

Mx-gate	generic identity	class
AND	MIN, algebraic AND	AND
OR	MAX, algebraic OR	OR
NOT	INVERT, algebraic COMPLEMENT	NOT
INTERSECT	set-theoretic AND	AND
UNION	set-theoretic OR	OR
COMPLEMENT	set-theoretic NOT	NOT
ANDe	existential AND	AND
ORe	existential OR	OR
NOTe	existential NOT	NOT
EXIST	existential acknowledgement	AND

To understand some of the proposed Mx-gates first requires the following definitions for "full set" (converse of the empty set), and the "atomizer function" (which decomposes a heterogeneous set so that no member of the resultant set is a set).

Full Set - A set denoted by $\underline{U}$ and composed of

a) all members of a given countable set

b) all subsets of that countable set excluding all its single member subsets.

Example 2.2-1

Assume the universe is the set {a, b,c,d}. Then (a) of the full set definition contributes the elements a, b, c, and d to the full set, (b) contributes {a,b}, {a,c}, {a,d}, {b,c}, {b,d}, {c,d}, {a,b,c}, {a,b,d}, {b,c,d}, {a,c,d}, and {a,b,c,d}. The full set is then $\underline{U}$ = {a,b,c,d,{a,b}, {a,c},{a,d},{b,c},{b,d},{c,d},{a,b,c}, {a,b,d},{b,c,d},{a,c,d},{a,b,c,d}}.

——End of Example 2.2-1——

Atomizer Function If S = $\{s_1, s_2, \ldots, s_m\}$, A(S) denotes the "atomizer" function on S. If $\underline{U}$ is S's full set, "~" is set difference, and e is an element (= irreducible set member = atom), let $\underline{u} \subsetneq \underline{U}$ ∃: $\{\underline{U} \sim \underline{u}\}$ contains no elements and $e \in \underline{u}$. Then, A(S) ≡ A:S ⟶ $\underline{u}$ is one-one.

Example 2.2-2

Let S = {1,3,{6,7,8},{},2,{4,{5, 9}},0} where in this example, ø denotes only the null value[9]. Then A(S) = {1, 3,6,7,8,ø,2,4,5,9,0} = {1,3,6,7,8,2,4, 5,9,0}. If $S \subseteq \underline{u}$, A(S) = S, where $\underline{u}$ is from the atomizer function definition.

——End of Example 2.2-2——

If X is a set containing the null set as a member, note that A(X) defines the null value given the null set operand.

The ten definitions for Mx operations, beginning at the bottom of this page, conform to the GFS. In those Definitions, X = $\{x_1, x_2, \ldots, x_n\}$ is the input set to a Mx-gate and $x_i \in X$.

The NOT(X) definition needs a totally ordered r_g whose least member behaves like a zero. Its next higher member must behave like a one, the next higher must behave like a two, etc.[9]

"~" is set subtraction in the COMPLEMENT definition.

$$\textbf{UNION}(X) \equiv \begin{cases} \text{union}(\{A(x_1)\},\{A(x_2)\},\ldots,\{A(x_n)\}) \text{ if } \forall x_i \in X,\ x_i \subseteq r_g \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{AND}(X) \equiv \begin{cases} \min(A(X)) \text{ if } \forall x_i \in X,\ x_i \subseteq r_g \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{INTERSECT}(X) \equiv \begin{cases} \text{intersection}(\{A(x_1)\},\{A(x_2)\},\ldots,\{A(x_n)\}) \\ \quad \text{if } \forall x_i \in X,\ x_i \subseteq r_g, \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{COMPLEMENT}(X) \equiv \begin{cases} \{r_g \sim \text{union}(\{A(x_1)\},\{A(x_2)\},\ldots,\{A(x_n)\})\} \\ \quad \text{if } \forall x_i \in X,\ x_i \subseteq r_g \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{NOT}(X) \equiv \begin{cases} \max(r_g) - A(X) \text{ if } \forall x_i \in X,\ x_i \subseteq r_g \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{OR}(X) \equiv \begin{cases} \max(A(X)) \text{ if } \forall x_i \in X,\ x_i \subseteq r_g \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{ORe}(X) \equiv \begin{cases} x_1 \text{ if } x_1 \neq \emptyset,\ x_2 = x_3 = x_4 = \ldots = \emptyset \text{ and } x_1 \subsetneq r_g, \\ x_2 \text{ if } x_2 \neq \emptyset,\ x_1 = x_3 = x_4 = \ldots = \emptyset \text{ and } x_2 \subsetneq r_g, \\ \quad\vdots \\ x_n \text{ if } x_n \neq \emptyset,\ x_{n-1} = x_{n-2} = \ldots \\ \qquad = x_{n+1} = x_{n+2} = \ldots = \emptyset, \text{ and } x_n \subsetneq r_g, \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{ANDe}(X) \equiv \begin{cases} X \text{ if } \forall x_i \in X,\ x_i \subsetneq r_g, \text{ and } x_i \neq \emptyset \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{NOTe}(X) \equiv \begin{cases} r_g \text{ if } x_1 = x_2 = \ldots x_n = \emptyset \\ \emptyset \text{ otherwise} \end{cases}$$

$$\textbf{EXIST}(X) \equiv \begin{cases} \max(r_g) \text{ if } \forall x_i \in X,\ x_i \subsetneq r_g \text{ and } x_i \neq \emptyset \\ \emptyset \text{ otherwise} \end{cases}$$

EXIST(X) is also written as "Ee(X)" to identify its class.

2.2.1 Properties of the Proposed Function Set Mx uses {·,+,',^,∧,∨,",&,¢, ∉,∃} as algebraic "connectives" corresponding to {AND,OR,NOT,NOT,INTERSECT,UNION, COMPLEMENT,ANDe,ORe,NOTe,Ee}, respectively. (Note the two "NOT"s in the latter set. The apostrophe-type connective is used only when all variables are 2-valued and range over the same set.)

The execution precedence for Mx-expressions is:

1) The usual execution ordering applies for parenthetic statements in Mx-expressions.

2) Within each parenthetic statement, the operation precedence is, ordered left-to-right: ',^,",∉,∃,·,+,∧,∨,&,¢ . (By this ordering, x'y ≠ x^y and x'y = (x^)y .)

Following are some F properties which may be useful in manipulating Mx expressions. More are in the original work on Mx[9].

1) ∀(x,y) ⊆ r_g for a given Mx-gate of function * where * ∈ {·,+,^,∧,∨,",&,¢, ∉,∃}, x*y = y*x (x'y as a non-unary operation is undefined).

2) ∀(x) ⊆ r_g for a given Mx-gate of function *, then:

   a) Where * ∈ {·,+,∧,∨,&}, x*x = x (idempotency). But when * ∈ {^,", ¢,∉,∃}, x*x ≠ x (x'x is undefined). Special case: when x = ∅ and * ∈ {·,+,∧,∨,&,¢,∃}, ∅*∅ = ∅ (null idempotency).

   b) When * ∈ {·,+,∧,∨,&,¢}, x*∅ = x* = x, and ∅ is an identity variable. But when * ∈ {^,",∉,∃}, x*∅ = x* ≠ x (x'∅ is undefined).

   c) When * ∈ {^,"}, (x*)* = (x*∅)*∅ = x** = x (unary closure). When * ∈ {·, +,^,∨,"} and x ranges over values only, (x*)* = x** = x. When * ∈ {·,+,^,',∧,∨,",&,¢}, x ranges over values only, and x≠∅, (x*)* = x** = x. When * ∈ {∉,∃}, x** ≠ x . Example: When r_g = {0,1,2,3,4} and x = 3, x++ = 3+ = 3, x·· = 3· = 3, x^^ = 1^ = 3, x∉∉ = ∅∉ = {0,1,2,3,4} (i.e., x∉∉ ≠ x), x∃∃ = 4∃ = 4 (i.e., x∃∃ ≠ x).

3) ∀(x,y,z) ⊆ r_g, x*(y*z) = (x*y)*z where * ∈ {·,+,∧,∨,&,∃} (associativity).

4) ∀(x_i) ⊆ {r_g~∅} for a given Mx-gate of function * where * ∈ {·,+,^,∧,∨,", ¢,∉}, x_1 * x_2 * ... * x_m * $\emptyset_{m+1}$ * $\emptyset_{m+2}$ * ... * $\emptyset_n$ = x_1 * x_2 * ... * x_m. (The subscripted nulls represent inputs each of which is currently null.)

5) Where x ≠ ∅ and r_g is not a single-member set, we observe:
   x∧x^ = ∅ x∧x" = ∅ x∨x" = {x,x"}

2.2.2 Completeness in Mx Completeness for the Mx GFS and for the F members is conditional[9]. Existing definitions of completeness (e.g., [6,8]) cannot be applied directly to F because they do not treat input alphabets and universes of operation

as independent from the operators. Definitions of completeness appropriate to Mx follow. "Moderate" completeness accompanies the notions of strong and weak completeness.[4]

Strong Completeness - $F_j \subseteq F$ in Mx is "strong complete" over a set S of two or more values iff the minimum count of iterations of networks (including the initial one) composed of members of F_j required to cover S is not less than the cardinality of S.

Moderate Completeness - $F_j \subseteq F$ in Mx is "moderate complete" over a set S having two or more values iff the minimum count of iterations of networks (including the initial one) composed of F_j members required for the union of their outputs to be identical to S is less than the cardinality of S.

Citizen - Any input instance which is a subset of r_g. "Citizen" is the antonym of "alien."

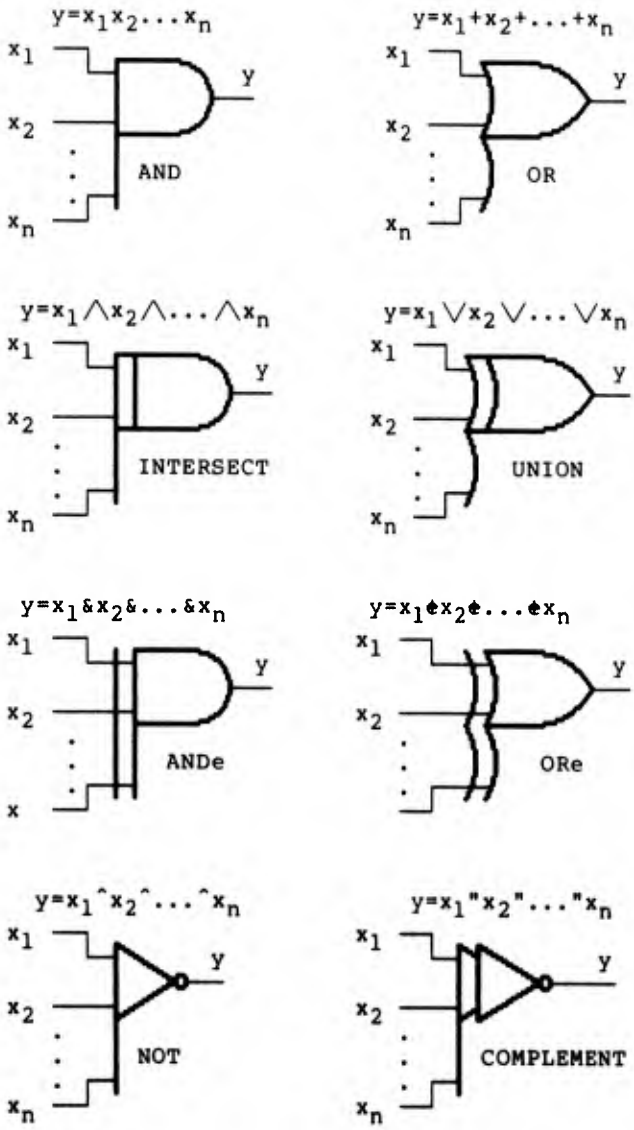
Weak Completeness - $F_j \subseteq F$ in Mx is "weak complete" with respect to a set S where $|S| \geq 2$ iff some citizen(s) of or alien(s) to the F_j members is(are) needed at any input(s) at any iteration of the networks in the strong or moderate complete definitions, excluding the initial iteration, in order to make F_j otherwise strong or moderate complete.

$F_j \subseteq F$ in Mx is moderate complete if also strong complete. The converse may not be true. A strong or moderate complete F_j is weak complete iff all input instances to its members also belong to S. If F_j is weak or moderate complete, it cannot be made stronger without replacing and/or including one or more functions. Any strong or moderate complete simple basis[4] requires the inclusion of at least one monadic function, or an n-adic function which gives a defined output with all its inputs connected together, to service single-output networks.*

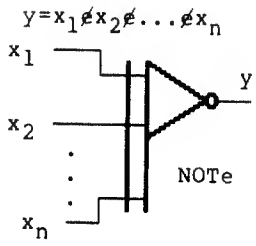
In the following list of predetermined[9] F member completeness, s, m, w, and i mean possibly strong, moderate, or weak completeness, and incomplete, respectively.

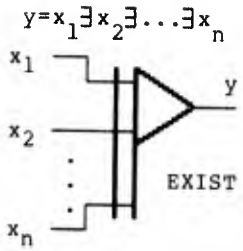
AND	i
OR	i
NOT	w, conditionally s or m
INTERSECT	i
UNION	i
COMPLEMENT	s or m
ANDe	i
ORe	i
NOTe	s or m
EXIST	i

2.2.3 Gate Symbol Graphics for the Proposed Function Set Symbols for AND, OR, and NOT gates are the usual shapes. The remaining symbols are variations on these.



* A monadic operation is defined as an operation requiring one input (argument). An n-ary operation is defined as an m-adic operation having n inputs, where $m \leq n$.



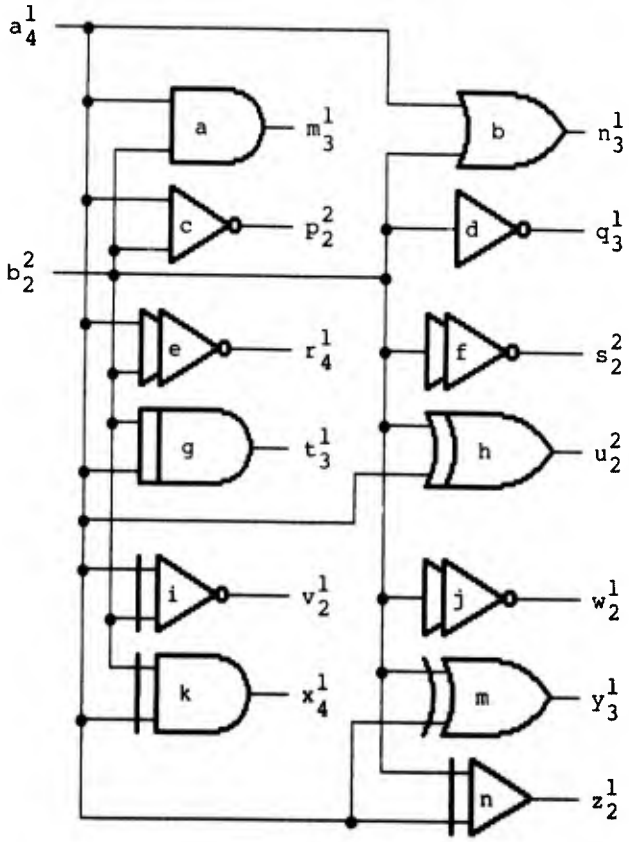


When only 2-valued variables exist and over the same set of values, "^" is unary; "'" may be used instead. "'" is undefined for non-unary applications of NOT.

2.2.4 Mx Gate Evaluations This section gives examples of Mx-gate behavior. A treatment of set I/O is included. Variables shown sub- and superscripted with radix and bus signal cardinality, respectively, specify bus realizations.

Example 2.2.4-1

Assume a $\in$ {ø,0,1} and b $\in$ {ø,0,1,2}. Note that the input buses are not used to their capacities. The input sets are obvious from the figure. Assume all gate reference sets to be r_g = {0,1} with 0<1.



Application of the proposed Mx-function Definitions yields the following truth tables. Note that all gates yield a null when an alien input is present.

a	b	m= ab	n= a+b	p = a^b	q= b^	r = a"b	s= b"	t= a∧b	u= a∨b
0	0	0	0	1	1	0	1	0	0
0	1	0	1	{0,1}	0	ø	0	ø	{0,1}
1	0	0	1	{0,1}	1	ø	1	ø	{0,1}
1	1	1	1	0	0	1	0	1	1
0	ø	0	0	1	ø	1	{0,1}	0	0
1	ø	1	1	0	ø	0	{0,1}	1	1
ø	0	0	0	1	1	1	1	0	0
ø	1	1	1	0	0	0	0	1	1
ø	ø	ø	ø	ø	ø	{0,1}	{0,1}	ø	ø
0	2	ø	ø	ø	ø	ø	ø	ø	ø
1	2	ø	ø	ø	ø	ø	ø	ø	ø
ø	2	ø	ø	ø	ø	ø	ø	ø	ø

a	b	v = a⊄b	w= b⊄	x = a&b	y = a⊕b	z = a∃b
0	0	ø	ø	0	ø	1
0	1	ø	ø	{0,1}	ø	1
1	0	ø	ø	{0,1}	ø	1
1	1	ø	ø	1	ø	1
0	ø	ø	{0,1}	ø	0	1
1	ø	ø	{0,1}	ø	1	1
ø	0	ø	ø	ø	0	1
ø	1	ø	ø	ø	1	1
ø	ø	{0,1}	{0,1}	ø	ø	1
0	2	ø	ø	ø	ø	ø
1	2	ø	ø	ø	ø	ø
ø	2	ø	ø	ø	ø	ø

Null outputs require physical representation (e.g., a D.C. potential) in practice even though ø $\in$ {no value, empty set, no answer/output, ...}.

If inputs are restricted to be {0,1}, the outputs of the COMPLEMENT and NOT Mx-gates are identical.

For a$\in${0,1}, b$\in${0,1}, and outputs t and u in the truth tables, if the null and set outputs are changed to 0 and 1 respectively, then the operations of AND and INTERSECT, and OR and UNION are indistinguishable.

The truth tables show by exhaustion that AND, OR, and NOT Mx-gates, under 0,1 inputs and {0,1} reference sets, yield results identical to 2-valued Boolean AND, OR, and NOT.

——End of Example 2.2.4-1——

Methods exist for representing and physically handling set I/O.[9] One possibility for outputs is to detect unique set instances within the Mx-gate, then deliver a set identifier to the external circuitry. For example, the symbol "5" can be assigned to a set such as {0,1,2}. A transmitted set identifier (e.g., "5") can be interpreted by receiving Mx-gates as desired.

This set identifier method can be used to "create" elements not in a transmitting gate's reference set. For example, if r_g =

{1,2,3}, then outputs of 4 = {1,2,3}, 5 = {1,2}, 6 = {2,3}, 7 = {1,3} can be "created" at its gate's output.

2.3 A Way to Design Using Mx

The following design procedure approximates the truth-table methods familiar to users of 2-valued Boolean algebra.

Procedure 2.3-1 - First-Order Logic Design Using Mx

1) Build a truth table for the problem having columns for its inputs followed by columns for its outputs.

2) Fill each row with an I/O instance so that all possible I/O conditions are covered.

3) For each non-ø input instance, determine (if any) a NOT class gate and reference set to yield it.

4) For each non-ø output, determine an AND class gate and reference set to yield it. If satisfactory gate and reference set pairs are all found, go to step 5. Otherwise, insert a blank column before the output columns.

   a) For each non-ø output, determine an AND class gate and reference set pair and test input (install in the new column) to yield it. If satisfactory input and gate and reference set pair combinations cannot all be found, keep the best ones, insert another blank column before the previous one, then go to step 4a. Otherwise, for each test input given the original inputs, determine an AND class gate and reference set pair to yield it.

5) For each output column and pair or other test tuples of rows having non-ø outputs, determine a solving cascade of OR class gate and reference set pairs. If satisfactory gate and reference set pairs are all found, go to step 6. Otherwise, append a blank column to the truth table.

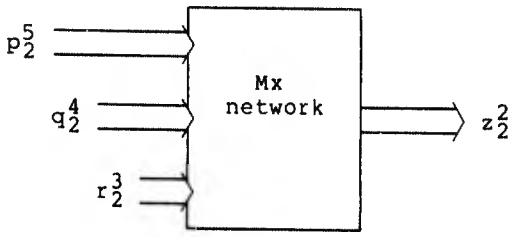
   a) Similar to step 4a, solve for the test inputs but using OR class gates. Failing, keep the best test sets, insert another blank column before the previous one, then go to step 5a.

6) Draw the circuit and/or write the equations accordingly.

——End of Procedure 2.3-1——

Procedure 2.3-1 does not promise a 2-level AND-OR solution. Especially in steps 3 through 5a, much room for interpretation and style exists, which is why the Procedure is called "First-Order." The following simple example uses Procedure 2.3-1. More substantial examples exist.[9]

Example 2.3-1



Specification: Transmit to output z the value from set {a,b,c} common to buses p, q, and r.

The bus carrying signal p has a physical capacity of 2^5 = 32 values, an example use of the notations. Output z is physically 4-valued: one for each member of {a,b,c} and one assigned to ø. (The super/subscripts have no algebraic significance and make the broad arrow bus symbol unnecessary.)

Steps 1 and 2 of Procedure 2.3-1 yield the following truth table.

p	q	r	z
a	a	a	a
b	b	b	b
c	c	c	c
otherwise			ø

Applying step 3, no "inversions" exist (e.g., a≠c^) for any input instance, so no NOT class gates are needed.

Applying step 4, three non-ø outputs exist, so three AND class gates are needed. Any of an AND, INTERSECT, or ANDe gate yields the value common to their inputs. (Two OR class gates, the OR and UNION, also do this.) Arbitrarily select three INTERSECT gates.

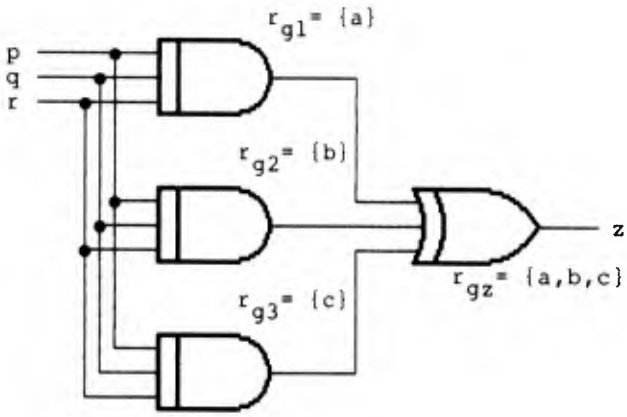
Value "a" is required at the output of the INTERSECT gate assigned to the row where all inputs are equal to "a"; otherwise, ø is. Assigning r_{g1}={a} to the gate accomplishes this. Similarly assign r_{g2}={b} and r_{g3}={c} to the INTERSECT gates for the rows where z=b and z=c, respectively.

By step 5, since no more than one AND class gate output will be non-ø at any time, any single 3-input OR class gate will

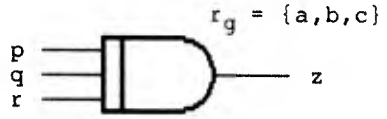
suffice. (Three AND class gates, the AND, INTERSECT, and ANDe would also serve as well.) UNION is arbitrarily chosen.

Only the values a, b, or c can appear at the inputs of the chosen UNION gate. Hence, it can have any reference set r_{gz} provided that {a,b,c} is a subset of it. Arbitrarily choose r_{gz}={a,b,c}.

Step 6 yields the following circuit, corresponding to the expression, z = (p∧q∧r)∨(p∧q∧r)∨(p∧q∧r).



One is tempted to apply idempotency to the expression for z, hoping to reduce it to z = p∧q∧r. Such an application of idempotency is generally invalid, since each gate has a different reference set. However, a single 3-input INTERSECT gate with r_g = {a,b,c} allows z = p∧q∧r, reducing the previous circuit to the following equivalent.



Had any AND-class gate other than INTERSECT been chosen, a single-gate solution would not have been possible. All other gates would produce undesired outputs given r_g = {a,b,c}, and no other reference set would have sufficed.

Intuition suggested using a different reference set to arrive at the single-gate equivalent. Procedure 2.3-1 yielded a correct result, but not one having minimum Mx-gate count. Noticing earlier that the truth table describes set-theoretic intersection under {a,b,c,} would have led directly to z = p∧q∧r, an application of the definition of INTERSECT.

——End of Example 2.3-1——

2.4 A Way To Turn Mx Designs Into Hardware

Hardware directly realizing the gates of F would be useful.* Two-valued Boolean algebra enjoys that position in small scale ICs (integrated circuits). Some means for realizing Mx-designs have been reported.[9] But one more method, table-lookup,** is simple yet exploits VLSICs (very large scale ICs) for modest size reference sets and input value cardinalities. Table-lookup, applied to the single-gate solution of Example 2.3-1 follows.

Assume the single-gate solution in Example 2.3-1, a 3-input INTERSECT gate, is to be built in a 2-valued technology as indicated by the Example's first diagram. Then, since input r is a 3-wide bus and the remaining two buses are wider, inputs p and q can be reduced to 3-wide buses also.

Assume that the values a, b, and c are represented by the symbols 2, 5, and 7, respectively, in binary arithmetic notation. The value range of interest must include the alphabet {2,5,7}. With that as the only constraint, arbitrarily choose p∈{0-7}, q∈{0-7}, and r∈{0-7}.

Two methods of constructing table-lookup hardware use PLAs (programmable logic arrays) or memories. We arbitrarily select an EPROM (erasable, programmable read-only memory).

Since the solution circuit has three 3-wide bus inputs, an EPROM having at least nine (3+3+3) address bits are needed. The output requires at least three bits for non-null representation. Adopting a convention where a null output is signified by the most significant bit being a binary one requires that the output be increased to at least four bits. This implementation requires two more output bits than suggested in Example 2.3-1 because that version assumed the minimal physical configuration. That is, for an n-valued output, no more than $\log_2 n$ signal lines are needed in a bus.

Since nine address bits and four output bits are required, an EPROM with 512 4-bit words (a "2K" EPROM) is needed; well within state-of-the-art technology. (Speed is not considered in this implementation.) Therefore, an EPROM programmed as shown in the following truth tables can directly realize the single gate solution to Example 2.3-1. (A generalization of this EPROM method can serve in other Mx-networks as a

---

* A patent has been initiated.

** Idea suggested by Dr. Richard K. Kunze, Dept. of Defense, 24 August

standard realization of an INTERSECT gate. This is known as a "standard cell" approach.[3])

p q r	z	address in decimal notation	output in binary notation
7 7 7	7	511	0111
5 5 5	5	365	0101
2 2 2	2	146	0010
other	ø	other	1xxx

3.0 Conclusions

A technology-independent, heterogeneous, mix-valued logic design algebra denoted by "Mx" was discussed. As an algebra intended for use mainly by logic design practitioners instead of only researchers, Mx is atypical. It has been applied where I/O consists of values and/or sets of values and/or variables. Function I/O appears attractive for preserving numeric representation accuracy.[5] As examples, a) The natural number, e, may be operated on as a token, saving numeric representation until the last moment, and b) Division, even though not in the proposed Mx-function set, may be carried through a network — a fraction such as 1/3 may be symbolized then decimally represented at the last moment, reducing losses in precision due to recursive operations.

The behavior of Mx-gates depends upon their reference sets (which may be different among the Mx-gates) and input alphabets (which may be different among the inputs) for algebraic properties. Reference sets can be varied with time or other parameters. Real-time changes to the nature of an entire hardware circuit may thereby be possible. Many algebraic properties of Mx remain to be explored.

Mx-gate-level minimization methods are as yet unexplored. Their availability could be useful to realization methods like the one discussed in Section 2.4.

Completeness, associativity, and distributivity in the proposed Mx function set are all conditional. Yet by Example 2.3-1 and others[9], Mx has been shown to be useful in the design of logic circuits having 2- or multi-valued signals and their buses.

References

1. BAER, J. L., "A Survey of Some Theoretical Aspects of Multiprocessing," Computing Surveys, vol. 5, No. 1, March 1973, pp. 31-80.

2. ENDERTON, Herbert B., A Mathematical Introduction to Logic, Academic Press, 1972, p. 9.

3. FRANK, Edward H., and SPROULL, Robert F., "Testing and Debugging Custom Integrated Circuits," Computing Surveys, Vol. 13, No. 4, December 1981, pp. 425-451.

4. MUKHOPADHYAY, Amar, "Complete Sets of Logic Primitives," Recent Developments in Switching Theory, Edited by Amar Mukhopadhyay, Academic Press, 1971, pp. 1-26.

5. PETERSON, Ivars, "Can You Count on Your Computer?," Science News, 31 July 1982, Vol. 122, No.5, pp. 72-75.

6. RESCHER, Nicholas, Many-valued Logic, McGraw-Hill Book Company, 1969.

7. RINE, David C., Editor, Computer Science and Multiple-Valued Logic, Theory and Applications, North-Holland, 1977.

8. ROSENBERG, Ivo G., "completeness properties of multiple-valued logic algebras," ibid., pp. 144-186.

9. SINUTKO, Michael Jr., "A Mix-Valued Algebra for Combinatorial Digital Logic Design," Ph.D. Dissertation, University of Maryland, College Park, Maryland, 1982.